ACADEMIC PRESS

Available at www.ComputerScienceWeb.com

POWERED BY SCIENCE DIRECT®

Information and Computation 183 (2003) 2–18

Information and Computation

www.elsevier.com/locate/ic

Analysing the implicit complexity of programs

J.Y. Marion

Loria Calligramme Project B.P. 239, 54506 Vandoeuvre-lès-Nancy Cedex, France

Abstract

We study termination proofs in order to (i) determine computational complexity of programs and (ii) generate efficient programs from the complexity analysis. For this, we construct a termination ordering, called *light multiset path ordering* (LMPO), which is a restriction of the *multiset path ordering*. We establish that the class of first order functional programs on lists which is terminating by LMPO characterises exactly the functions computable in polynomial time.

1. Introduction

What is the complexity of a function computed by a program? How can an efficient program which computes the same function be extracted? We shall attempt to answer to those questions. We consider first order functional programs whose data structure are lists. The result is a syntactic delineation of a significative class of programs which defines exactly all functions running in polynomial time. It is difficult to ascribe a precise meaning to "significative" because we are dealing with intentional properties of programming languages. So, we mean a class of programs containing algorithms which solve nontrivial problems naturally, as we shall try to illustrate in Example 4.2. The result is based on an analysis of program termination proofs to determine an approximation of the complexity of the computed function that we call *the analysis of the implicit complexity of a program*. We shall see that the programs considered calculate in exponential time, but denote polynomial time functions. For this, we combine termination orderings for term-rewriting systems (see the survey of Dershowitz [8]) and the ramification of recurrence schema arising from the work of Simmons [18], Bellantoni and Cook [2], and Leivant [15]. A similar approach, which replaces the above ramification principle by a semantic argument, has been

E-mail address: Jean-Yves.Marion@loria.fr.
URL: http://www.loria.fr/∼marionly.

0890-5401/03/$ - see front matter © 2003 Published by Elsevier Science (USA).
doi:10.1016/S0890-5401(03)00011-7

suggested by Marion and Moyen [16]. Compared to the analysis of the complexity of a program like Benzinger's prototype [3], this approach allows us to transform programs. The transformation is based on dynamic programming methods which were developed first by Cook [6] and generalised by Andersen and Jones [1,12].

This paper is organised as follows. Section 2 defines functions computed by rewrite systems and multiset path ordering. The construction of light multiset path orderings is given in Section 3. In Section 4, the main result is presented in Theorem 18. The proof is established by Theorems 19 and 20. The proof of Theorem 20 is first outlined and details are in Sections 5 and 6.

2. Programming with rewrite rules

2.1. Programs over many-sorted algebra

A *signature* is a pair $(\mathcal{S}, \Sigma)$ where $\mathcal{S}$ is a finite set of sorts (atomic types) and Σ is a vocabulary on $\mathcal{S}$. A vocabulary on $\mathcal{S}$ consists in a finite set of symbols. Every symbol in Σ has some fixed arity n and a type. The type of a symbol of arity 0 is a sort s of $\mathcal{S}$. The type of a symbol of arity $n > 0$ is an $(n+1)$-uplet of sorts of $\mathcal{S}$ that we write $\mathsf{s}_1, \ldots, \mathsf{s}_n \to \mathsf{s}$.

Throughout, for every sort $\mathsf{s} \in \mathcal{S}$ there is a countably infinite set χ_s of variables of sort s which is disjoint from vocabularies. Put $\chi = \bigcup_{\mathsf{s}\in\mathcal{S}} \chi_\mathsf{s}$.

The set $\mathcal{T}(\Sigma, \chi)_\mathsf{s}$ is the set of freely generated terms of type s. The set of terms is $\mathcal{T}(\Sigma, \chi) = \bigcup_{\mathsf{s}\in\mathcal{S}} \mathcal{T}(\Sigma, \chi)_\mathsf{s}$. The set of ground terms of type s is denoted by $\mathcal{T}(\Sigma)_\mathsf{s}$. The set of ground terms is $\mathcal{T}(\Sigma) = \bigcup_{\mathsf{s}\in\mathcal{S}} \mathcal{T}(\Sigma)_\mathsf{s}$. The set of variables in a term t is $\mathrm{Var}(t) \subseteq \chi$. A ground substitution is a mapping σ from χ to $\mathcal{T}(\Sigma)$, which respects type variables. The size, $|t|$, of a term $t \in \mathcal{T}(\Sigma, \chi)$ is the number of symbols in t

$$|t| = \begin{cases} 1 & \text{if } t \text{ is 0-ary or a variable,} \\ \sum_{i=1}^{n} |t_i| + 1 & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

Definition 1. A program is a quadruplet $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ where
(a) $\mathcal{S}$ is a set of sorts.
(b) $\mathcal{C}$ is a vocabulary on $\mathcal{S}$ and symbols of $\mathcal{C}$ are called constructors.
(c) $\mathcal{F}$ is a vocabulary on $\mathcal{S}$, disjoint of $\mathcal{C}$, and symbols of $\mathcal{F}$ are of arity > 0 and are called function symbols.
(d) $\mathcal{E}$ is a finite sequence of oriented equations. Every equation is of the form $f(t_1, \ldots, t_n) \to t$ where $f \in \mathcal{F}$ is of type $\mathsf{s}_1, \ldots, \mathsf{s}_n \to \mathsf{s}$, each $t_i \in \mathcal{T}(\mathcal{C}, \chi)_{\mathsf{s}_i}$, $t \in \mathcal{T}(\mathcal{C} \cup \mathcal{F}, \chi)_\mathsf{s}$ such that $FV(t) \subseteq \cup_{i \leqslant n} \mathrm{Var}(t_i)$.

The main function symbol is the last symbol defined in $\mathcal{E}$.

2.2. Semantics

Actually, a program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ denotes a term rewriting system. The set of equations $\mathcal{E}$ induces a rewriting rule $\to$ defined as follows: $u \to v$ if the term v is obtained from u by applying an

equation of $\mathcal{E}$. The relation $\xrightarrow{+}$ ($\xrightarrow{*}$) denotes the transitive (reflexive-transitive) closure of $\rightarrow$. We write $s \xrightarrow{!} t$ to mean that $s \xrightarrow{*} t$ and t is in normal form. A program is confluent if the induced rewriting rule $\rightarrow$ is confluent. One might consult [9] about general references on rewrite systems.

We give an operational semantics based on term rewriting. The domain of computation of a program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ is the term algebra defined by the signature $\langle \mathcal{S}, \mathcal{C} \rangle$. The interpretation of the sort s is the ground term algebra built on constructor terms; that is, $[\![\mathsf{s}]\!] = \mathcal{T}(\mathcal{C})_{\mathsf{s}}$.

Definition 2. Let F be the main symbol of a confluent program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ of type $\mathsf{s}_1, \ldots, \mathsf{s}_n \rightarrow \mathsf{s}$. The function computed by $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ is $[\![F]\!] : [\![\mathsf{s}_1]\!] \times \overset{1}{\cdots} \times [\![\mathsf{s}_n]\!] \mapsto [\![\mathsf{s}]\!]$ which is defined as follows. For all $u_i \in [\![\mathsf{s}_i]\!]$, $[\![F]\!](u_1, \ldots, u_n) = v$ iff $f(u_1, \ldots, u_n) \xrightarrow{1} v$ and $v \in [\![\mathsf{s}]\!]$, otherwise $[\![F]\!](u_1, \ldots, u_n)$ is undefined.

Throughout, we shall only consider the normal forms, which are in $\mathcal{T}(\mathcal{C})$, as being meaningful.

Example 3. The following equations define the function which returns the minimum value in a list of numbers. The sorts are $\mathcal{S} = \{\mathsf{Nat}, \mathsf{List}(\mathsf{Nat})\}$ and constructors are $\mathcal{C} = (\mathbf{0} : \mathsf{Nat}, \mathbf{suc} : \mathsf{Nat} \rightarrow \mathsf{Nat}, \mathbf{nil} : \mathsf{List}(\mathsf{Nat}), \mathbf{cons} : \mathsf{Nat},\ \mathsf{List}(\mathsf{Nat}) \rightarrow \mathsf{List}(\mathsf{Nat})\}$.

$$\mathsf{inf} : \mathsf{Nat}, \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$\mathsf{inf}(\mathbf{0}, y) \rightarrow \mathbf{0}$$
$$\mathsf{inf}(x, \mathbf{0}) \rightarrow \mathbf{0}$$
$$\mathsf{inf}(\mathbf{suc}(x), \mathbf{suc}(y)) \rightarrow \mathbf{suc}(\mathsf{inf}(x, y))$$
$$\mathsf{listInf} : \mathsf{List}(\mathsf{Nat}) \rightarrow \mathsf{Nat}$$
$$\mathsf{listInf}(\mathbf{nil}) \rightarrow \mathbf{0}$$
$$\mathsf{listInf}(\mathbf{cons}(n, l)) \rightarrow \mathrm{inf}(n, \mathsf{listInf}(l))$$

Putting $\mathsf{n} = \mathbf{suc}^n(0)$, we have $[\![\mathsf{inf}]\!](\mathsf{n}, \mathsf{m}) = \min(\mathsf{n}, \mathsf{m})$. The program listInf denotes $[\![\mathsf{listInf}]\!]$ $(\mathbf{cons}(\mathsf{n}_1, \ldots, \mathbf{cons}(\mathsf{n}_p, \mathbf{nil}))) = \min_{i \leqslant \mathsf{p}}(\mathsf{n}_i)$.

2.3. Multiset path ordering

Multiset path ordering (MPO) was introduced by Plaisted [17] and Dershowitz [7]. MPO belongs to the family of syntactic term orderings whose well-foundness stems on Kruskal's tree theorem [14].

We briefly describe MPO. A multiset M of terms of $\mathcal{T}(\Sigma, \chi)$ is a finite mapping $M : \mathcal{T}(\Sigma, \chi) \mapsto \mathbb{N}$ which associates to each term t the number $M(t)$ of terms t in M. Suppose that $\mathcal{T}(\Sigma, \chi)$ is ordered by $\prec$. This ordering $\prec$ induces an ordering $\prec^{\mathrm{m}}$ on term multisets.

Definition 4. $M \prec^{\mathrm{m}} N$ iff $M \neq N$ and for all $s \in \mathcal{T}(\Sigma, \chi)$ if $M(s) > N(s)$, then there is $t \in \mathcal{T}(\Sigma, \chi)$ such that $s \prec t$ and $M(t) < N(t)$.

Throughout, the subterm relation will be noted $\lhd$. For example, we see that $\{s(x), x, x\} \lhd^{\mathrm{m}} \{s(x), s(x), x\}$.

Let $\prec_\Sigma$ be an ordering on Σ and $\approx_\Sigma$ be an equivalence relation on Σ which respects symbol arities. Then, we say that the quasi-ordering $\preceq_\Sigma$ defined by $\prec_\Sigma \cup \approx_\Sigma$ is a precedence on Σ. From $\approx_\Sigma$, we define the *permutative congruence* $\approx$ as the smallest equivalence relation on terms which satisfied $f(t_1,\ldots,t_n) \approx g(s_1,\ldots,s_n)$ if $f \approx_\Sigma g$ and $t_i \approx_{S_{\pi(i)}}$ for some permutation π over $\{1,\ldots,n\}$.

Definition 5. Let $\preceq_\Sigma$ be a precedence on the signature Σ. The multiset path ordering $\prec_{\mathrm{mpo}}$ is defined recursively by

1. $s \prec_{\mathrm{mpo}} f(\ldots,t_i,\ldots)$, if $s \prec_{\mathrm{mpo}} t_i$
2. $g(s_1,\ldots,s_m) \prec_{\mathrm{mpo}} f(t_1,\ldots,t_n)$ if
   (a) either $g \prec_\Sigma f$ and $s_i \prec_{\mathrm{mpo}} f(t_1,\ldots,t_n)$ for all $i \leqslant m$,
   (b) or $g \approx_\Sigma f$ and $\{s_1,\ldots,s_m\} \prec^{\mathrm{m}}_{\mathrm{mpo}} \{t_1,\ldots,t_n\}$,

where $\preceq_{\mathrm{mpo}} = \prec_{\mathrm{mpo}} \cup \approx$.

A program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}\rangle$ is terminating by MPO if there is a precedence on $\mathcal{C} \cup \mathcal{F}$ such that for each equation $l \to r$ of $\mathcal{E}$, we have $r \prec_{\mathrm{mpo}} l$. Notice that we do take into account the types of terms to prove termination.

Hofbauer has shown in [10] that the class of functions computed by a confluent program which is terminated by MPO is exactly the class of the primitive recursive functions.

Example 6. The following program computes the exponential and terminates by MPO.$\mathsf{d} : \mathsf{Nat} \to \mathsf{Nat}$ and $[\![\mathsf{d}]\!](\mathsf{n}) = 2 \cdot \mathsf{n}$

$$\mathsf{d}(\mathbf{0}) \to \mathbf{0}$$
$$\mathsf{d}(\mathbf{suc}(x)) \to \mathbf{suc}(\mathbf{suc}(\mathsf{d}(x)))$$

$\mathsf{exp} : \mathsf{Nat} \to \mathsf{Nat}$ and $[\![\mathsf{exp}]\!](\mathsf{n}) = 2^{\mathsf{n}}$

$$\mathsf{exp}(\mathbf{0}) \to \mathbf{suc}(\mathbf{0})$$
$$\mathsf{exp}(\mathbf{suc}(x)) \to \mathsf{d}(\mathsf{exp}(x))$$

3. Light multiset path ordering

3.1. Valency and extension of orderings to n-uplets

The definition of a light multiset path ordering makes use of the notion of valency to restrict MPO.

Definition 7. A valency of a function symbol f of arity n is a mapping $v(f) : \{1,\ldots,n\} \mapsto \{0,1\}$.

We shall write $f(\ldots,t_i,\ldots)$ to mean that the term t_i occurs at position i in f, which is a position of valency $v(f,i)$.

The valency of a function symbol f indicates how to compare arguments of f. More precisely, it indicates first how to compare n-uplets of arguments and second how to compare an argument with a term by giving the ability of combining several term orderings. Thus, valencies generalise the notion of status as proposed by Kamin and Levy [13] in the following way. The status of function symbols allow one to compare $f(s_1,\ldots,s_n)$ with $f(t_1,\ldots,t_n)$ by lifting the term ordering to an ordering on sequences of terms which depends on f. The notion of valency extends this idea to the case where root symbols are different Throughout, we shall assign valencies to the function symbols $\mathcal{F}$, but we shall never assign valencies to constructors of $\mathcal{C}$. Constructors of $\mathcal{C}$ will be always treated separately.

Definition 8. Let v be a valency function on $\mathcal{F}$. A permutation π over $\{1,\ldots,n\}$ respects the valency of f and g if the arity of f and g is n and $v(g,i)=v(f,\pi(i))$ for all $i\leqslant n$.

Definition 9. Let $\prec_0$ and $\prec_1$ be two term orderings. These orderings are lifted to an ordering over n-uplets of terms which respects the valency function v on $\mathcal{F}$ as follows. $\{s_1,\ldots,s_n\}\prec^v_{g,f}\{t_1,\ldots,t_n\}$ iff there is a permutation π which respects the valency of f and g and which satisfies that there is $j\leqslant n$ such that $v(g,j)=1$ and $s_j\prec_1 t_{\pi(j)}$, and for all $i\leqslant n$, $s_i\preceq_{v(g,i)} t_{\pi(i)}$.

The above ordering on n-uplets of terms is a restriction of the multiset ordering, as defined in Definition 4, induced by the transitive closure of $\prec_1\cup\prec_0$.

Definition 10. Let $\approx_{\mathcal{F}}$ be an equivalence relation on $\mathcal{F}$. The *permutative congruence* $\approx$ *which respects the valency* v is the smallest equivalence relation on terms of $\mathcal{T}(\mathcal{C}\cup\mathcal{F},\chi)$ which satisfies:
1. $\mathbf{c}(s_1,\ldots,s_n)\approx\mathbf{c}(t_1,\ldots,t_n)$ if $\mathbf{c}\in\mathcal{C}$ and $s_i\approx t_i$ for all $i\leqslant n$.
2. $f(t_1,\ldots,t_n)\approx g(s_1,\ldots,s_n)$ if $f\approx_{\mathcal{F}} g, t_i\approx s_{\pi(i)}$ for some permutation π which respects the valency of f and g.

Definition 11. Let $\preceq_{\mathcal{F}}$ be a precedence on $\mathcal{F}$. *The light multiset path ordering* is a pair $(\prec_k)_{k=0,1}$ of orderings which is recursively defined on $\mathcal{T}(\mathcal{C}\cup\mathcal{F},\chi)$ as follows:
1. $s\prec_k\mathbf{c}(\ldots,t_i,\ldots)$ if $s\preceq_k t_i$ and $\mathbf{c}\in\mathcal{C}$.
2. $s\prec_k f(\ldots,t_i,\ldots)$ if $s\preceq_k t_i, f\in\mathcal{F}$, and $k\leqslant v(f,i)$.
3. $\mathbf{c}(s_1,\ldots,s_n)\prec_k f(t_1,\ldots,t_m)$ if $\mathbf{c}\in\mathcal{C}, f\in\mathcal{F}$, and $s_i\prec_k f(t_1,\ldots,t_m)$, for each $i\leqslant n$. Note that $\mathbf{c}$ can be a 0-ary.
4. $g(s_1,\ldots,s_n)\prec_k f(t_1,\ldots,t_m)$ if $(g\prec_{\mathcal{F}} f)$ and if $s_i\prec_{\max(k,v(g,i))} f(t_1,\ldots,t_m)$ for each $i\leqslant n$.
5. $g(s_1,\ldots,s_n)\prec_0 f(t_1,\ldots,t_n)$ if $g\approx_{\mathcal{F}} f$ and $\{s_1,\ldots,s_n\}\prec^v_{g,f}\{t_1,\ldots,t_n\}$,

where $\preceq_k=\prec_k\cup\approx$.

3.2. Properties

Proposition 12.
(a) $\preceq_0$ *is an extension of* $\preceq_1$; *that is,* $s\preceq_1 t$ *implies* $s\preceq_0 t$.
(b) *The multiset path ordering* $\preceq_{\mathrm{mpo}}$ *is an extension of* $\preceq_0$; *that is,* $s\preceq_0 t$ *implies* $s\preceq_{\mathrm{mpo}} t$.

Proof. Each rule of $\preceq_1$ is a rule of $\preceq_0$. By forgetting function symbol valencies, we also see that each rule of $\preceq_0$ is also a rule of $\preceq_{\mathrm{mpo}}$. □

Lemma 13. *The relation* $(\prec_i)_{i=0,1}$ *is a partial ordering.*

Proof. Since $\prec_{\mathrm{mpo}}$ is an extension of $\prec_i$, we immediately see that $\prec_i$ is irreflexive.

The transitivity of $\prec_i$ is tedious to demonstrate. So, we just outline the key steps of the proof. Assume that $s \preceq_k t$. We have the following:

1. If $s = \mathbf{c}(s_1, \ldots, s_n)$ and $\mathbf{c} \in \mathcal{C}$, then for each $i \leqslant n, s_i \prec_k t$.
2. If $s = g(s_1, \ldots, s_n)$ and $g \in \mathcal{F}$, then for each $i \leqslant n, s_i \prec_{\max(k,v,(g,i))} t$.
3. If $\mathbf{c} \in \mathcal{C}$, then $s \prec_k \mathbf{c}(\ldots, t, \ldots)$.
4. If $v(f,i) = k$, then $s \prec_k f(\ldots, t, \ldots)$. where t is the ith argument of f.

We prove these four assertions by a mutual induction on $|s| + |t|$. From then on, we can prove the transitivity of $\preceq_i$ again by induction on $|s| + |t|$. □

Neither $\preceq_1$ nor $\preceq_0$ is a total ordering. They are not simplification orderings because they are not monotonic. However, note that $\preceq_0$ is monotonic with respect to the argument of valency 1; i.e., $s_i \preceq_1 t_i$ implies $f(\ldots, s_i, \ldots) \preceq_0 f(\ldots, t_i, \ldots)$ if $v(f,i) = 1$. Also $\prec_0$ possesses the subterm property; that is, $t \prec_0 f(\ldots, t, \ldots)$.

Proposition 14. *Let* s *and* t *be two terms of* $\mathcal{T}(\mathcal{C})$. *We have* $s \preceq_1 t$ *iff* $s \preceq_0 t$ *iff* $s \trianglelefteq t$.

Proof. (1) $s \preceq_1 t$ implies $s \preceq_0 t$ by Proposition 12. (2) To prove that $s \preceq_0 t$ implies $s \trianglelefteq t$ we proceed by induction on $|t|$. But first observe that $s \approx t$ is equivalent to $s = t$ by Definition 10. Otherwise $s \prec_0 t = \mathbf{c}(t_0, \ldots, t_n)$, necessarily $s \preceq_0 t_j$, and so $s \trianglelefteq t_j \triangleleft t$. (3) $s \trianglelefteq t$ implies $s \preceq_0 t$ is proved by induction on $|t|$. □

4. Characterisation of ptime

4.1. LMPO programs

Definition 15. A LMPO *program* is a sextuple $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$, where $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$ is a confluent program, v is a valency function on $\mathcal{F}$, $\preceq_{\mathcal{F}}$ is a precedence on $\mathcal{F}$, and such that the induced ordering $\prec_0$ satisfies for each rule $l \to r, r \prec_0 l$.

Termination of LMPO programs is obtained by the fact that the term ordering $\prec_0$ is a restriction of $\prec_{\mathrm{mpo}}$, and so the rewrite relation of a LMPO program is well founded.

4.2. Examples

We shall now give several examples. This example shows that the class of LMPO programs encapsulates a broad class of algorithmic patterns.

For the sake of readability, we write the termination proof in sequent style. On the right, we note the rule number which is applied, following Definition 11. Also, we write in subscript the valency of arguments of a function symbol in the declaration of its type. Last we use a notational convention similar to that of [2] and write $f(x_1, \ldots, x_n; y_1, \ldots, y_m)$, with a semicolon separating two

lists of arguments, to indicate that $v(f,i)=1$ for $i\in\{1,\ldots,n\}$ and $v(f,n+j)=0$ for $j\in\{1,\ldots,m\}$.

1. The program displayed in Example 3 is a LMPO program where valencies are set as $v(\mathsf{inf},1)=1$, $v(\mathsf{inf},2)=0$, $v(\mathsf{listInf},1)=1$. The precedence is $\mathsf{inf}\prec_{\mathcal{F}}\mathsf{listInf}$. The two first equations are ordered because $\prec_0$ possesses the subterm properties. The termination proof of the third one is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x\approx x}{x\prec_1\mathbf{suc}(x)}(1)\quad\dfrac{y\approx y}{y\prec_0\mathbf{suc}(y)}(1)}{\{x,y\}\prec^{\nu}_{\mathsf{inf}}\{\mathbf{suc}(x),\mathbf{suc}(y)\}}(\text{Def. 9})}{\mathsf{inf}(x;y)\prec_0\mathsf{inf}(\mathbf{suc}(x);\mathbf{suc}(y))}(5)}{\mathbf{suc}(\mathsf{inf}(x;y))\prec_0\mathsf{inf}(\mathbf{suc}(x);\mathbf{suc}(y))}(3)$$

The fourth equation is a special case of (3). The termination proof of the last equation is carried out as follows.

$$\dfrac{\dfrac{\dfrac{n\approx n}{n\prec_1\mathbf{cons}(n,l)}(1)}{n\prec_1\mathsf{listInf}(\mathbf{cons}(n,l);)}(2)\quad\dfrac{\dfrac{l\approx l}{l\prec_1\mathbf{cons}(n,l)}(1)}{\mathsf{listInf}(l;)\prec_0\mathsf{listInf}(\mathbf{cons}(n,l);)}(5)\quad\mathsf{inf}\prec_{\mathcal{F}}\mathsf{listInf}}{\mathsf{inf}(n;\mathsf{listInf}(l))\prec_0\mathsf{listInf}(\mathbf{cons}(n,l);)}(4)$$

The program inf computes the minimum of two integers n and m, written in unary, within $O(\min(n,m))$ steps. So inf is the "good" algorithms for the function $[\![\mathsf{inf}]\!]$. Colson has demonstrated in [5] that if we consider the class of programs defined by means of strict primitive recursion, then the function $[\![\mathsf{inf}]\!]$ is obviously definable but not with a "good" algorithm. In particular, this implies that the characterization of the polynomial time functions of Bellantoni and Cook [2] does not comprise a "good" algorithm for $[\![\mathsf{inf}]\!]$, due to the absence of simultaneous recursion is that class.

2. Given a list *list* of type List(Nat), sort(*list*, length(*list*);) returns the *list* sorted. The algorithm is the insertion sort. The sort bool, with (**tt**: bool, **ff**: bool) as a constructor set, denotes the truth values.

$\mathsf{if\_then\_else\_}:\mathsf{bool}_0,\mathsf{Nat}_0,\mathsf{Nat}_0\to\mathsf{Nat}$

$\quad\mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ x\ \mathsf{else}\ y\to x$

$\quad\mathsf{if}\ \mathbf{ff}\ \mathsf{then}\ x\ \mathsf{else}\ y\to y$

$<:\mathsf{Nat}_1,\mathsf{Nat}_0\to\mathsf{bool}$

$\quad\mathbf{0}<\mathbf{suc}(y)\to\mathbf{tt}$

$\quad x<\mathbf{0}\to\mathbf{ff}$

$\quad\mathbf{suc}(x)<\mathbf{suc}(y)\to x<y$

$\mathsf{length}:\mathsf{List}(\mathsf{Nat})_1\to\mathsf{Nat}$

$$\mathsf{length}(\mathbf{nil}) \to 0$$

$$\mathsf{length}(\mathbf{cons}(a,l)) \to \mathbf{suc}(\mathsf{length}(l))$$

$$\mathsf{insert} : \mathsf{Nat}_1, \mathsf{Nat}_1, \mathsf{List}(\mathsf{Nat})_0 \to \mathsf{List}(\mathsf{Nat})$$

$$\mathsf{insert}(0,a;l) \to \mathbf{cons}(a,l)$$

$$\mathsf{insert}(\mathbf{suc}(n),a;\mathbf{cons}(b,l)) \to \mathsf{if}\ a < b\ \mathsf{then}\ \mathbf{cons}(a,\mathbf{cons}(b,l))\ \mathsf{else}\ \mathbf{cons}(b,\mathsf{insert}(n,a;l))$$

$$\mathsf{sort} : \mathsf{List}(\mathsf{Nat})_1, \mathsf{Nat}_1 \to \mathsf{List}(\mathsf{Nat})$$

$$\mathsf{sort}(\mathbf{nil},0;) \to \mathbf{nil}$$

$$\mathsf{sort}(\mathbf{cons}(a,l),\mathbf{suc}(n);) \to \mathsf{insert}(n,a;\mathsf{sort}(l,n;))$$

Put $\mathsf{if\_then\_else\_} \prec_{\mathcal{F}} \_ < \_ \prec_{\mathcal{F}} \mathsf{insert} \prec_{\mathcal{F}} \mathsf{sort}$. Then, each equation is ordered by $\prec_0$. Take, for example, the last equation,

$$\dfrac{\dfrac{\dfrac{n \approx n}{n \prec_1 \mathbf{cons}(n,l)}(1)}{n \prec_1 T}(1) \quad \dfrac{\dfrac{a \approx a}{a \prec_1 \mathbf{cons}(a,l)}(1)}{a \prec_1 T}(1) \quad \dfrac{\dfrac{n \approx n}{n \prec_1 \mathbf{suc}(n)}(1) \quad \dfrac{l \approx l}{l \prec_1 \mathbf{cons}(a,l)}(1)}{\mathsf{sort}(l,n;) \prec_0 T}(4)}{\mathsf{insert}(n,a;\mathsf{sort}(l,n;)) \prec_0 T}(4)$$

where $T = \mathsf{sort}(\mathbf{cons}(a,l),\mathbf{suc}(n);)$ The extra parameter in sort which denotes the length of the list is necessary because it comes from banning the use of critical arguments such as $\mathsf{sort}(l,n;)$ as recurrence parameters. Hofmann [11] allows definition by recurrence over a nonincreasing function by defining a type system which counts constructor applications. Also, Caseiro [4] has designed some semantic conditions on term rewriting systems to permit such recursion.

3. In Example 6, the exponential function was defined. The rules concerning d are ordered by $\prec_0$. In particular, the second rule forces the valency of d to be 1. As a consequence, the fourth rule cannot be ordered by $\prec_0$ because $\exp(x) \not\preceq \exp(\mathbf{suc}(x))$.

4. Given two strings $u = u_1, \ldots, u_m$ and $v = v_1, \ldots, v_n$ of {0,1}*, a common subsequence of length k is defined by two sequences of indices $i_1 < \cdots < i_k$ and $j_1 < \cdots < j_k$ satisfying $u_{i_q} = v_{j_q}$. Consider the well-known problem which consists of computing the longest common subsequence of two words. To write a LMPO program to solve this problem, we encode binary words by the sort word generated by the constructors $\{\epsilon : \mathsf{word}, \mathbf{1} : \mathsf{word} \to \mathsf{word}, \mathbf{2} : \mathsf{word} \to \mathsf{word}\}$. Then we write the recursive solution of the problem.

$$\mathsf{max} : \mathsf{word}_1, \mathsf{Nat}_0, \mathsf{Nat}_0 \to \mathsf{Nat}$$

$$\mathsf{max}(x;n,\mathbf{0}) \to n$$

$$\mathsf{max}(x;\mathbf{0},m) \to m$$

$$\mathsf{max}(\mathbf{i}(x);\mathbf{suc}(n),\mathbf{suc}(m)) \to \mathbf{suc}(\mathsf{max}(x;n,m)) \quad \mathbf{i} \in \{\mathbf{1},\mathbf{2}\}$$

$$\mathsf{lcs} : \mathsf{word}_1, \mathsf{word}_1 \to \mathsf{word}$$

$$\mathsf{lcs}(x, \epsilon;) \to \mathbf{0}$$

$$\mathsf{lcs}(\epsilon, y;) \to \mathbf{0}$$

$$\mathsf{lcs}(\mathbf{i}(x), \mathbf{i}(y);) \to \mathbf{suc}(\mathsf{lcs}(x, y;))$$

$$\mathsf{lcs}(\mathbf{i}(x), \mathbf{j}(y);) \to \mathsf{max}(\mathbf{i}(x); \mathsf{lcs}(x, \mathbf{j}(y);), \mathsf{lcs}(\mathbf{i}(x), y;)) \quad \mathbf{i} \neq \mathbf{j}$$

The program is ordered by $\prec_0$ by putting $\mathsf{max} \prec_{\mathcal{F}} \mathsf{lcs}$.

The recursive definition of $[\![\mathsf{lcs}]\!]$ has an exponential growth. In particular, this means that the normal form is obtained within an exponential number of term reductions. In fact, each recursive solution of a problem which is solved in polynomial time by dynamic programming, and which is ordered by MPO, might be analysed similarly.

4.3. Simple constructor sets

Definition 16. A signature $\langle \mathcal{S}, \mathcal{C} \rangle$ is simple if for every constructor $\mathbf{c}$ of type $\mathsf{s}_1, \ldots, \mathsf{s}_n \to \mathsf{s}$, the sort s appears at most once in $\mathsf{s}_1, \ldots, \mathsf{s}_n$.

Therefore, the sort s appears at most once in $\{\mathsf{s}_1, \ldots, \mathsf{s}_n\}$. Throughout, we shall consider programs which run over a data-structure based on simple constructor signatures. Booleans (bool), tally numbers (Nat), words (e.g., word), and lists (e.g., List(Nat)) are simple constructor signatures. However, trees are not generated by simple constructor sets because a constructor of type $\mathsf{Btree} \times \mathsf{Btree} \to \mathsf{Btree}$ is at least necessary.

The main motivation behind the introduction of simple constructors is that the size of a term is polynomially bounded in its height The height, $\mathrm{ht}(t)$, of a term $t \in \mathcal{T}(\mathcal{C})$ is defined thus

$$\mathrm{ht}(t) = \begin{cases} 1 & \text{if } t \text{ is a 0-ary constructor,} \\ \max_{i=1}^{n} \mathrm{ht}(t_i) + 1 & \text{if } t = \mathbf{c}(t_1, \ldots, t_n). \end{cases}$$

Proposition 17. *Let $\langle \mathcal{S}, \mathcal{C} \rangle$ be a simple signature and d be the maximal arity of a constructor of $\mathcal{C}$. For each term $t \in \mathcal{T}(\mathcal{C})$ whose type is of level k, we have $|t| \leqslant d^k \cdot \mathrm{ht}(t)^{k+1}$.*

Proof. The proof goes by induction on the level of the sort and on the size of the term. Suppose that the level is 0. Each constructor is at most unary, and so $|t| \leqslant \mathrm{ht}(t)$. Suppose that $t = \mathbf{c}(t_1, \ldots, t_n)$ and is of level $k+1$. Since the signature is simple, we know that there is at most one subterm t_p which is of level $k+1$. From the inductions hypothesis, we have $|t| \leqslant 1 + \sum_{i \neq p} d^{k-1} \cdot \mathrm{ht}(t_i)^k + d^k \cdot \mathrm{ht}(t_p)^{k+1}$. Since $n \leqslant d$, we have $|t| \leqslant 1 + d^k \cdot \max_{i \leqslant n} \mathrm{ht}(t_i)^{k+1}$. The conclusion follows because $\mathrm{ht}(t_i) < \mathrm{ht}(t)$. □

4.4. Main result

The notion of valency is a generalisation of the idea of data-tiering, as in [2] and [15]. This differs principally from the tiering-systems considered up to now in the following respects.

Constructors have no valency, i.e., are not tiered, and the notion of valency is applied to the analyses of a broader class of programs.

Theorem 18. *Each LMPO program over a simple constructor signature is computable in polynomial time, and conversely each polynomial time function is computed by a LMPO program over a simple constructor signature.*

Proof. This is a consequence of Theorems 19 and 20. □

Theorem 19. *Each function ϕ computable in polynomial time is represented by a LMPO program.*

Proof. We represent each function of the class B, as it was introduced in [2], by a LMPO program. Then, the desired conclusion will follow immediately because the class B is exactly the class of polynomial time computable functions. The domain is represented by the sort word. The positions of normal inputs will be of valency 1, and the positions of safe inputs will be of valency 0. Each initial function and schema of the class B is straightforwardly simulable by a LMPO program. For example, take the safe recursion schema:

$$f : \mathsf{word}_1, \mathsf{word}_1, \mathsf{word}_0 \to \mathsf{word}$$
$$f(\epsilon, x; a) \to g(x; a)$$
$$f(\mathbf{i}(y), x; a) \to h_i(y, x; a, f(y, x; a))$$

By putting $g, h_0, h_1 \prec_{\mathcal{F}} f$, we check that $h_i(y, x; a, f(y, x; a)) \prec_0 f(\mathbf{i}(y), x; a)$, because $y, x \prec_1 f(\mathbf{i}(y), x; a)$ and $f(y, x; a) \prec_0 f(\mathbf{i}(y), x; a)$.

So, for each B function $\phi : \mathbb{N}^n \times \mathbb{N}^m \mapsto \mathbb{N}$, we can write a program of LMPO such that $\forall u_1, \ldots, u_n, v_1, \ldots, v_m \in \mathcal{T}$ (word), we have

$$\tau([\![f(u_1, \ldots, u_n, v_1, \ldots, v_m)]\!]) = \phi(\tau(u_1), \ldots, \tau(u_n); \tau(v_1), \ldots, \tau(v_m)),$$

where $\tau(\epsilon) = 0$ and $\tau(\mathbf{i}(x)) = 2x + i$. □

Theorem 20. *Let $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$ be a LMPO program of the main symbol F. For all inputs $a_1, \ldots, a_r \in \mathcal{T}(\mathcal{C})$, the computation of $[\![F(a_1, \ldots, a_r)]\!]$ is time bounded by $p(\max_{i \leqslant r} |a_i|)$ where p is some polynomial.*

Proof. The next sections are devoted to the theorem's proof. The difficulty to calculate a LMPO program is illustrated by Example 4.2(4). Indeed, the length of term derivations as well as the size of terms involved can be exponential in the size of arguments. We design a call-by-value evaluation procedure which memorises values of subcomputations. This is described in Section 6. The algorithm is similar to that used by Jones in [12] to show that a class of read-only programs with while-loop and recursion is computable in polynomial time. The argument is a rereading of Cook's simulation [6] of two-way multihead pushdown automata in polynomial time. The evaluation procedure transforms a program based on rewrite rules into a faster and equivalent one.

The time bound is obtained by two arguments. First, we provide in Section 5 a quasi-polynomial interpretation on the height of terms involved in a derivation. Since the height and

the size of simple constructors are polynomially bounded, we obtain that the size of normal form is bounded by a polynomial in the size of the inputs (Theorem 30). But, this upper-bound is not sufficient to prove the time bound. Second, at the end of Section 6, we establish that the number of recursive calls is also bounded by a polynomial in the size of the arguments. □

5. Bounding normal form size

Fix a program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$. Without loss of generality, we assume that $\preceq_{\mathcal{F}}$ is total. Define $\mathcal{F}_1, \ldots \mathcal{F}_k$ as the partition of $\mathcal{F}$ determined by $\preceq_{\mathcal{F}}$ such that $g \in \mathcal{F}_q$ and $f \in \mathcal{F}_{q+1}$ iff $g \prec_{\mathcal{F}} f$, and $f \approx_{\mathcal{F}} g$ iff f and g are in $\mathcal{F}_q$. We say that if f is in $\mathcal{F}_q$, then f is of *rank* q. Intuitively, we consider constructors as symbols of rank 0. We write $\#S$ to denote the cardinal of the set S.

5.1. A polynomial interpretation of terms

We define a sequence of polynomials parameterised by some constant $d \geqslant 2$ as follows,

$$F_0(X) = X^d, \tag{1}$$

$$F_{k+1}(X) = F_k^d(X), \tag{2}$$

where $F_k^\alpha(X)$ means α iterations of F_k, i.e., $F_k(\ldots(F_k(X))\ldots)$. Throughout, *$d$ will stand for the parameter which defined the sequence (F_k) and $d \geqslant 2$.*

Definition 21. The interpretation $[\,]$ is defined on $\mathcal{T}(\mathcal{C} \cup \mathcal{F})$ as follows:
- $[\mathbf{b}] = d$ for every 0-ary constant $\mathbf{b}$ of $\mathcal{C}$.
- $[\mathbf{c}(t_1, \ldots, t_n)] = \max_{i \leqslant n}[t_i] + d$ for every constructor $\mathbf{c}$ of $\mathcal{C}$.
- $[f(t_1, \ldots, t_n)] = F_{k+1}(\max(d, \sum_{v(f,p)=1}[t_p])) + \max_{v(f,i)=0}[t_i]$, for every $f \in \mathcal{F}$ of rank $k+1$.

Remark 22. $[f(t_1, \ldots, t_n)] = F_{k+1}(\sum_{v(f,p)=1}[t_p]) + \max_{v(f,i)=0}[t_i]$ if f has at least one argument of valence 1. Otherwise, if f has no argument of valence 1, then $[f(t_1, \ldots, t_n)] = F_{k+1}(d) + \max_{v(f,i)=0}[t_i]$.

It is immediate to see from Definition 10 that if $s \approx t$ then $[s] = [t]$. Also, we have $\mathrm{ht}(t) \leqslant [t]$, for each term t.

5.2. Properties of F_k

We study some properties on the sequence (F_k) that we shall use as lemmas.

Proposition 23. *For all X, k, and d,*
1. $F_k(X) = X^{d^{d^k}}$
2. $F_{k+1}^\alpha(X) = F_k^{\alpha \cdot d}(X)$
3. *For all $j \leqslant k$ and $X \leqslant Y$, we have $X \leqslant F_j(X) \leqslant F_k(Y)$.*

Proof. (1) and (2) are proved by showing by induction on (k,α) that we have $F_k^\alpha(X) = X^{d^{\alpha \cdot d^k}}$. (3) is a consequence of (1). □

Proposition 24. *For all k, and $d \geqslant 2$, we have*
1. *For all $X \geqslant d$ and $\alpha, \beta > 0, F_k^\alpha(X) + F_k^\beta(X) \leqslant F_k^{\alpha+\beta}(X)$*
2. *For all X and $\alpha < d, F_k^\alpha(X+1) + F_{k+1}(X) \leqslant F_{k+1}(X+1)$.*

Proof. (1) By induction on k. Case $k = 0$; assume that $\alpha \geqslant \beta$. We have $X^{d^\alpha} + X^{d^\beta} \leqslant 2.X^{d^\alpha} \leqslant X^{d^{(\alpha+1)}}$ because $X \geqslant d$, and so $\leqslant X^{d^{(\alpha+\beta)}}$. Case $k < 0$,

$$\begin{aligned} F_{k+1}^\alpha(X) + F_{k+1}^\beta(X) &= F_k^{d\alpha}(X) + F_k^{d\beta}(X) \quad \text{(by Proposition 23(2))} \\ &\leqslant F_k^{d\cdot(\alpha+\beta)}(X) \quad \text{(by induction hypothesis)} \\ &= F_{k+1}^{\alpha+\beta}(X) \quad \text{(by Proposition (23(2)).} \end{aligned}$$

(2) Assume that $\gamma < \beta$; we have $(X+1)^\gamma + X^\beta \leqslant (X+1)^\beta$, because $(X+1)^\beta = (X+1)(X+1)^{\beta-1} \geqslant X^\beta + (X+1)^{\beta-1}$. Since $\alpha < d$, we have $d^{d^k\alpha} < d^{d^{k+1}}$. We conclude by replacing γ by $d^{d^k\alpha}$ and β by $d^{d^{k+1}}$ in the former inequality. □

5.3. Bounding theorem

Lemma 25. *Let s and $t = f(t_1, \ldots, t_n)$ be two terms of $\mathcal{T}(\mathcal{C} \cup \mathcal{F}, \chi)$ such that $Var(s) \subseteq Var(t)$ and $s \prec_1 t$.*

Assume also that for all $q \leqslant n$ and for all terms u, if $u \prec_1 t_q$ then for each ground substitution $\sigma, [u\sigma] \leqslant [t_q\sigma]$.

Take a ground substitution σ. Then, $[s\sigma] \leqslant F_k^{|s|}(A)$ where f is a function symbol of rank $k+1$ and $A = \max(d, \sum_{v,(f,q)=1}[t_q\sigma])$.

Proof. The proof is by inductions on $|s|$.

Assume $|s| = 1$. Suppose that s is a 0-ary constructor of $\mathcal{C}$. We have $[s\sigma] = d \leqslant A \leqslant F_k(A)$ by Proposition 23(3). Suppose that s is a variable of χ. Since $s \preceq_1 t$, it is necessary that $s \preceq_1 t_q$, for some q satisfying $v(f,q) = 1$ and $s \in \mathrm{Var}(t_q)$. By the hypothesis of the lemma, $[s\sigma] \leqslant [t_q\sigma] \leqslant A$. So, $[s\sigma] \leqslant F_k(A)$ by Proposition 23(3).

Assume $|s| > 1$. Suppose that $s \preceq_1 t_q$ and that $v(f,q) = 1$. By the hypothesis of the lemma, $[s\sigma] \leqslant [t_q\sigma] \leqslant A$. Hence, $[s\sigma] \leqslant F_k(A)$ by Proposition 23(3). Suppose that $s = \mathbf{c}(s_1, \ldots, s_m)$ where $\mathbf{c} \in \mathcal{C}$ and that for each $p \leqslant m, s_p \preceq_1 t$.

$$\begin{aligned} [s\sigma] &\leqslant \max_{p \leqslant m} F^{|s_p|}(A) + d \quad \text{(by induction hypothesis)} \\ &\leqslant \max_{p \leqslant m} F^{|s_p|}(A) + F_k(A) \quad (d \leqslant A \leqslant F_k(A)) \\ &\leqslant F^{1\sum_{p \leqslant m}|s_p|}(A) \quad \text{(by Proposition 24(1)).} \end{aligned}$$

Suppose that $s = f_\rho(s_1, \ldots, s_m)$ where f_ρ is a function symbol of $\mathcal{F}$ of rank $\rho \leqslant k$. For each $p \leqslant m$, $s \preceq_1 t$. We have

$$\begin{aligned}
[s\sigma] &\leqslant F_\rho\left(\sum_{p\leqslant m}[s_p\sigma]\right) \quad \text{(by definition of [])}\\
&\leqslant F_\rho\left(\sum_{p\leqslant m}F_k^{|s_p|}(A)\right) \quad \text{(by induction hypothesis)}\\
&\leqslant F_\rho\left(F_k^{\sum_{p\leqslant m}|s_p|}(A)\right) \quad \text{(by Proposition 24(1))}\\
&\leqslant F_k^{|s|}(A) \quad \text{(by Proposition 23(3)).} \qquad \square
\end{aligned}$$

Lemma 26. *Let s and $t = f(t_1, \ldots, t_n)$ be two terms of $\mathcal{T}(\mathcal{C} \cup \mathcal{F}, \chi)$ such that $Var(s) \subseteq Var(t)$. Suppose that $s \prec_0 t$.*

Assume also, that for all $j \leqslant n$ and all terms u, if $u \prec_0 t_j$ then for each ground substitution σ, $[u\sigma] < [t_j\sigma]$.

Take a ground substitution σ. We have

$$[s\sigma] \leqslant F_k^{|s|}(A) + F_{k+1}(A-1) + B, \tag{3}$$

where $A = \max(d, \sum_{v,(f,q)=1}[t_q])$, $B = \max_{v(f,j)=0}[t_j\sigma]$, and f is a function symbol of rank $k+1$.

Proof. We proceed by induction on $|s|$.

Assume $|s| = 1$. Suppose that s is the 0-ary constructor of $\mathcal{C}$. Since $[s\sigma] = d \leqslant A$, so (3) holds. Suppose that s is a variable. Since $s \prec_0 t$, it is necessary that $s \preceq_0 t_j$ for some j such that $s \in \mathrm{Var}(t_j)$. By the hypothesis of the lemma, $[s\sigma] \leqslant [t_j\sigma] \leqslant A + B$, so (3) holds.

Assume $|s| > 1$. Suppose that $s \preceq_0 t_j$ for some j. By the lemma assumptions, we have $[s\sigma] \leqslant [t_j\sigma] \leqslant A + B$, so (3) holds. Suppose that $s = \mathbf{c}(s_1, \ldots, s_m)$ where $\mathbf{c} \in \mathcal{C}$ and that for each $i \leqslant m, s_i \preceq_0 t$. We have

$$\begin{aligned}
[s] &\leqslant \max_{p\leqslant m}F_k^{|s_p|}(A) + F_{k+1}(A-1) + B + d \quad \text{(by induction hypothesis)}\\
&\leqslant \max_{p\leqslant m}F_k^{|s_p|}(A) + F_{k+1}(A-1) + B + F_k(A) \quad (d \leqslant A \leqslant F_k(A))\\
&\leqslant F_k^{|s|}(A) + F_{k+1}(A-1) + B \quad \text{(by Proposition 24(1)).}
\end{aligned}$$

Suppose that $s = f_\rho(s_1, \ldots, s_m$ where $f_\rho \in \mathcal{F}$ is of rank $\rho \leqslant k$ and that for each $i \leqslant m$, $s_i \prec_{v(f_\rho,i)} t$. We can apply Lemma 25 because of the lemma assumptions and because $\prec_0$ is an extension of $\prec_1$ by Proposition 12(a). So, for each p such that $v(f_\rho, p) = 1$, we have $[s_p\sigma] \leqslant F_k^{|s_p|}(A)$. So, we have $\max(d, \sum_{v(f_\rho,p)=1}[s_p\sigma]) \leqslant \sum_{v(f_\rho,p)=1}F_k^{|s_p|}(A)$. It follows

$$F_\rho\left(\max\left(d,\sum_{v(f_\rho,p)=1}[s_p\sigma]\right)\right)\leqslant F_\rho\left(\sum_{v(f_\rho,p)=1}F_k^{|s_p|}(A)\right)\quad\text{by 23(3)}$$

$$\leqslant F_\rho\left(\sum_{v(f_\rho,p)=1}F_k^{|s_p|}(A)\right)\quad(A\geqslant d)$$

$$\leqslant F_\rho\left(F_k^{\sum_{v(f_\rho,p)=1}|s_p|}(A)\right)\quad\text{(by Proposition 24(1))}$$

$$\leqslant F_k^{1+\sum_{v(f_\rho,p)=1}|s_p|}(A)\quad\text{(by Proposition 23(3)).}$$

By definition, $[s\sigma]=F_\rho(\max(d,\sum_{v(f_\rho,p)=1}[s_p\sigma]))+\max_{v(f_\rho,i)=0}[s_i\sigma]$. By replacing with the above inequality, we have

$$[s\sigma]\leqslant F_k^{1+\sum_{v(f_\rho,p)=1}|s_p|}(A)+\max_{v(f_\rho,i)=0}[s_i\sigma].$$

Now, by induction hypothesis on arguments of valence 0,

$$[s\sigma]\leqslant F_k^{1+\sum_{v(f_\rho,p)=1}|s_p|}(A)+\max_{v(f_\rho,i)=0}F_k^{|s_i|}(A)+F_{k+1}(A-1)+B.$$

By Proposition 24(1), we see that

$$F_k^{1+\sum_{v(f_\rho,p)=1}|s_p|}(A)+\max_{v(f_\rho,i)=0}F_k^{|s_i|}(A)\leqslant F_k^{|s|}(A)$$

and so we obtain the inequality (3). Suppose that $s=g(s_1,\ldots,s_m)$ where $g\approx_\Sigma f$. Necessarily, there is a permutation π such that $s_i\preceq_{v,(g,i)}t_{\pi(i)}$ and $v(g,i)=v(f,\pi(i))$. The lemma assumptions yield $[s_i\sigma]\leqslant[t_{\pi(i)}\sigma]$. Moreover, there is p such that $v(g,p)=1$ and $s_p\prec_1 t_{\pi(p)}$. It follows by the lemma assumptions that $[s_p\sigma]<[t_{\pi(p)}\sigma]$. Consequently, $\sum_{v(g,p)=1}([s_p\sigma])<\sum_{v(f,q)=1}([t_q\sigma])=A$. So $\sum_{v(g,p)=1}([s_p\sigma])\leqslant A-1$. We obtain that $[s\sigma]\leqslant F_{k+1}(A-1)+B$. □

Theorem 27. *Let s and t be two terms of $\mathcal{T}(\mathcal{C}\cup\mathcal{F},\chi)$ such that $|s|<d$ and $s\prec_0 t$. Then, for every ground substitution σ, $[s\sigma]<[t\sigma]$.*

Proof. The proof goes by induction on $|t|$. Suppose that $t=\mathbf{c}(t_1,\ldots,t_n)$ where $\mathbf{c}$ is a constructor of $\mathcal{C}$. We have $s\preceq_0 t_q$. So $[s\sigma]\leqslant[t_q\sigma]$ by induction hypothesis. Then, it is immediate that $[s\sigma]<\max_{q\leqslant n}[t_q\sigma]+d$. Suppose that $t=f(t_1,\ldots,t_n)$ where f is of rank $k+1$. We can apply Lemma 26 by induction hypothesis. As previously, we put $A=\max(d,\sum_{v(f,q)=1}[t_q])$ and $B=\max_{v(f,j)=0}[t_j\sigma]$. So, we obtain $[s\sigma]\leqslant F_k^{|s|}(A)+F_{k+1}(A-1)+B$. By Proposition 24(2), we have $F_k^{|s|}(A)+F_{k+1}(A-1)<F_{k+1}(A)$. Therefore, we conclude $[s\sigma]<F_{k+1}(A)+B=[f_{k+1}(t_1,\ldots,t_n)\sigma]$. □

5.4. Quasi-interpretation of derivations

Lemma 28. *If* $[s] \leqslant [t]$ *then* $[f(\ldots, s, \ldots)] \leqslant [f(\ldots, t, \ldots)]$.

Theorem 29. *Let* $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$ *be a LMPO program. For each* $t, s \in \mathcal{T}(\mathcal{C} \cup \mathcal{F})$ *satisfying* $t \xrightarrow{*} s$, *we have* $[s] \leqslant [t]$.

Proof. Choose d in the interpretation [] such that for each rule $l \to r$, we have $d > |r|$. The proof is by induction on the length of the derivation by applying Lemma 28 and Theorem 27. □

Corollary 30. *Let* $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$ *be a LMPO program. For each* f *in* $\mathcal{F}$ *and each* $v_0, \ldots, v_n$ *in* $\mathcal{T}(\mathcal{C})$, *satisfying* $f(v_1, \ldots, v_n) \xrightarrow{!} v_0$, *we have* $|v_0| \leqslant p(\max_{i=1,n} |v_i|)$ *for some polynomial* p.

Proof. The definition of [] yields a polynomial q such that $[f(v_1, \ldots, v_n)] \leqslant q(\max_{i \leqslant n}[v_i])$. The theorem above yields $[v_0] \leqslant p(\max_{i \leqslant n}[v_i])$. Now $\mathrm{ht}(v_0) \leqslant [v_0]$. By Proposition 17, we conclude that $|v_0| \leq p(\max_{i \leqslant n} |v_i|)$, for some polynomial p. □

6. LMPO programs are ptime computable

We describe the evaluation procedure EVAL of LMPO programs. Fix a LMPO program $(\mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}})$. EVAL (t, σ) returns the normal form in $\mathcal{T}(\mathcal{C})$ of $t\sigma$, where $t \in \mathcal{T}(\mathcal{C} \cup \mathcal{S}, \chi)$ and σ is a store. A store is an assignation of the variables of t to a, value in $\mathcal{T}(\mathcal{C})$.

The computation of EVAL uses a global array G, called a minimal function graph (MFC) following the terminology of [12, Chap. 24.2] which is defined as follows. Say that a configuration is a couple $(g(u_1, \ldots, u_n), v)$ where g is a function symbol of $\mathcal{F}$, each u_i, is a constructor term of $\mathcal{T}(\mathcal{C})$, and which satisfies $g(u_1, \ldots, u_n) \xrightarrow{!} v$). A MFG is a set which contains configurations. The *MFG* G stores intermediate results to avoid costly recomputation.

The procedure EVAL works as follows. Initially the MFC G is empty. The procedure EVAL(t, σ) computes $t\sigma$ by call-by-value. When a term $t = g(u_1, \ldots, u_n)$ is considered where each $u_i\sigma$ is in $T(\mathcal{C})$, we search for a configuration $(g(u_1, \ldots, u_n)\sigma, v)$ in G. If it exists, we use it to short-cut the computation and so we return v. Otherwise, we apply a program equation, say $l \to r$, by matching $g(u_1, \ldots, u_n)$ with l through a store θ. Then we recursively compute $v = \text{EVAL}(r, \theta)$ and we add the configuration $(g(u_1, \ldots, u_n)\sigma, v)$ to G. The coherence of G is maintained since $g(u_1, \ldots, u_n)$ $\sigma \xrightarrow{+} r\theta \xrightarrow{!} v$. The procedure EVAL is detailed in Fig. 1.

It remains to estimate the time required by EVAL. We shall first establish that the size of G is polynomially bounded in the size of the input argument. Define G_p as the set of n-uplets of $\mathcal{T}(\mathcal{C})$ terms which are the arguments of function calls of rank p. That is, $(u_1, \ldots, u_n) \in G_p$ iff $(g(u_1, \ldots, u_n), v) \in G$ and the rank of g is p. We have

$$\#G \leqslant \sum_{1 \leqslant p \leqslant k} \#G_p, \tag{4}$$

```
Procedure EVAL
Inputs : t ∈ T(C ∪ F, X), a store σ
1     if tσ ∈ T(C) then return tσ
2     if t = c(h_1, ..., h_n) & c ∈ C then
3       for i = 1 to n do H_i := EVAL(h_i, σ)
4       return c(H_1, ..., H_n)
5     if t = f(h_1, ..., h_n) & f ∈ F then
6       for i = 1 to n do H_i := EVAL(h_i, σ)
7       if (f(H_1, ..., H_n), v) ∈ G then return v
8         else let θ and l → r be such that f(H_1, ..., H_n) = lθ
9              v = EVAL(r, θ)
10             G = G ∪ {(f(H_1, ..., H_n), v)}
```

Fig. 1. Evaluation of a LMPO program $\langle \mathcal{S}, \mathcal{C}, \mathcal{F}, \mathcal{E}, v, \preceq_{\mathcal{F}} \rangle$, given a minimal function graph G.

where k is the maximal rank of the function symbols in $\mathcal{F}$.

To give an upper-bound on $\#G_p$, we define, recursively on p, two sets $G_p^\vee$ and $G_p^\wedge$ thus. Consider the computation $\text{EVAL}(r, \theta)$ which is executed after applying the rule $l \to r$ with the store θ. Suppose that $l\theta = f(v_1, \ldots, v_m)$. Note that $v_i \in \mathcal{T}(\mathcal{C})$. Take a subterm $g(u_1, \ldots, u_n)$ of $r\theta$ where g is of rank p. We consider consider two cases.

(i) If the rank of f is strictly greater than p, then $(u'_1, \ldots, u'_n) \in G_p^\vee$ where $u_i \xrightarrow{!} u'_i$ for each $i < n$. It follows that the cardinality of $G_p^\vee$ is bounded by

$$\#G_p^\vee \leqslant \sum_{p < i \leqslant k} \#G_i. \tag{5}$$

(ii) Otherwise g and f are both of rank p; i.e., $g \approx f$ and so $n = m$. Since $g(u_1, \ldots, u_n) \prec_0 f(v_1, \ldots, v_n$, there is a permutation π such that $u_i \preceq_{v(g,i)} v_{\pi(i)}$. It follows from Proposition 14 that u_i is a subterm of $v_\pi(i)$; i.e., $u_i \trianglelefteq v_{\pi(i)}$. Hence, we write $(u_1, \ldots, u_n) \trianglelefteq^{\text{m}} (v_1, \ldots, v_n)$. (In fact, $\trianglelefteq^{\text{m}}$ is the extension of $\trianglelefteq$ as defined in 9.) We put $(u_1, \ldots, u_n) \in G_p^\wedge$.

The former observation means that all subcomputations of the rank p involved in the computation of $f(v_1, \ldots, v_n)$ have arguments which are subterms of the v_i's and so belong to

$$I(v_1, \ldots, v_n) = \{(u_1, \ldots, u_n) : (u_1, \ldots, u_n) \trianglelefteq^{\text{m}} (v_1, \ldots, v_n)\}$$

and so,

$$\#G_p^\wedge \leqslant \#\mathcal{F}_p \sum_{(v_1, \ldots, v_n) \in G_p^\wedge} \#I(v_1, \ldots, v_n).$$

But, $\#I(v_1, \ldots, v_n) \leqslant n! \cdot \sum_{i \leqslant n} |v_i|$. Suppose that $a_1, \ldots, a_n$ are the inputs. Then, Corollary 30 implies that there is a polynomial p such that for each $(v_1, \ldots, v_n) \in G_p^\vee, \sum_{i \leqslant n} |v_i| \leqslant n \cdot p(\max_{i \leqslant r} |a_i|)$. We obtain that

$$\#G_p^\wedge \leqslant \#G_p^\vee \cdot p'(\max_{i \leqslant r} |a_i|), \tag{6}$$

where the polynomial is $p'(x) = \#\mathcal{F}_p \cdot n! \cdot n \cdot p(x)$.

Finally, we have

$$\#G_p \leqslant \#G_p^\vee + \#G_p^\wedge. \tag{7}$$

By combining (4)–(7), we see that the cardinality of G is polynomially bounded in the size of the inputs. Since each value stored in G has a size bounded by $p(\max_{i\leqslant r}|a_i|)$ by Corollary 30, we conclude that the size of G is polynomially bounded in the input size.

Now, say that a description of EVAL is a quadruplet (l, G, t, σ) where l is a label of the algorithm of EVAL, G is a MFG, t is a term of $\mathcal{T}(\mathcal{C}\cup\mathcal{F},\chi)$, and σ is a store. The computation of EVAL can be described as a sequence of distinct descriptions, and the runtime is exactly the length of the sequence. Actually, the length of the description sequence depends on the size of the MFC G because (i) at each step, the size of G is identical or increasing, (ii) t is a subterm of an equation of the current LMPO program, and (iii) σ assigns values which are subterms of values in G or of the inputs. Now, we have established that the size of G is bounded by some polynomial in the size of the inputs. Consequently, we conclude that the evaluation algorithm EVAL terminates within $q(\max_{i\leqslant r}(|a_i|))$ steps, for some polynomial q. This completes the proof of Theorem 20.

References

[1] N. Andersen, N.D. Jones, Generalizing Cook's transformation to imperative stack programs, in: J. Karhumäki, H. Maurer, G. Rozenberg (Eds.), Results and Trends in Theoretical Computer Science, Lecture Notes in Computer Science, vol. 812, Springer-Verlag, Berlin/New York, 1994, pp. 1–18.

[2] S. Bellantoni, S. Cook, A new recursion-theoretic characterization of the poly-time functions, Comput. Complexity 2 (1992) 97–110.

[3] R. Benzinger, Automated Complexity Analysis of NUPRL Extracts, Ph.D. thesis, Cornell University, 1999.

[4] V.-H. Caseiro, An Equational Characterization of the Poly-Time Functions on Any Constructor Data Strucure,'' Technical Report 226, University of Oslo, Department of Informatics, 1996, available at http://www.ifi.uio.no/~ftp/publications.

[5] L. Colson, Functions versus algorithms, Bull. EATCS 65, the logic in computer science column, 1998.

[6] S. Cook, Characterizations of pushdown machines in terms of time-bounded computers, J. Assoc. Comput. Mach. 18 (1) (1971) 4–18.

[7] N. Dershowitz, Orderings for term-rewriting systems, Theoret. Comput. Sci. 17 (3) (1982) 279–301.

[8] N. Dershowitz, Termination of rewriting, J. Symbolic Comput. (1987) 69–115.

[9] N. Dershowitz, J.P. Jouannaud, Rewrite systems, in: Handbook of Theoretical Computer Science, vol. B, Elsevier Science, North-Holland, Amsterdam, 1990, pp. 243–320.

[10] D. Hofbauer, Termination proofs with multiset path orderings imply primitive recursive derivation lengths, Theoret. Comput. Sci. 105 (1) (1992) 129–140.

[11] M. Hofmann, Linear types and non-size-increasing polynomial time computation, in: 14th IEEE Symposium on Logic in Computer Science, 1999, pp. 464–473.

[12] N. Jones, Computability and Complexity, from a Programming Perspective, MIT Press, Cambridge, MA, 1997.

[13] S. Kamin, J.J. Lévy, Attempts for Generalising the Recursive Path Orderings, Technical report. University of Illinois, Urbana, Unpublished note, 1980.

[14] J.B. Kruskal, Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, Trans. Amer. Math. Soc. 95 (1960) 210–225.

[15] D. Leivant, Predicative recurrence and computational complexity I: Wont recurrence and poly-time, in: P. Clote, J. Remmel (Eds.), Feasible Mathematrics II, Birkhäuser, Basel, 1994, pp. 320–343.

[16] J.-Y. Marion, J.-Y. Moyen, Efficient first order functional program interpreter with time bound certifications, in: LPAR, Lecture Notes in Artificial Intelligence, vol. 1955, Springer-Verlag, New York, 2000, pp. 25–42.

[17] D. Plaisted, A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems, Technical Report R-78-943, Department of Computer Science, University of Illinois, 1978.

[18] H. Simmons, The realm of primitive recursion, Arch. Math. Logic 27 (1988) 177–188.